The Architecture of eBay Search

Andrew Trotman
University of Otago
andrew@cs.otago.ac.nz

Jon Degenhardt
eBay inc.
jdegenhardt@ebay.com

Surya Kallumadi
Kansas State University
surya@ksu.edu

ABSTRACT

The architecture of a large-scale search engine is dependent on the application. In the case of a web search engine, the document collection can be considered to be a constantly growing, but slowly changing, archive. The task is to find and crawl good pages and to index them. The rate of change of these pages can be estimated and the pages re-crawled and re-indexed periodically. Users issue queries, and the results to those queries are likely to be the same from day to day so can be cached.

eCommerce, on the other hand, is quite different. The document collection can change very quickly, the results of queries may differ from user to user, and crawling may not be required.

In this contribution, we outline the architecture of Cassini, the eBay search engine. eBay tackles a problem quite different from that of a traditional web search and consequently chose to design and build a search engine from scratch and to customize it to the nature of the problem.

CCS CONCEPTS

• **Information systems → Environment-specific retrieval**;

KEYWORDS

Product Search, Industrial Information Retrieval.

ACM Reference format:
Andrew Trotman, Jon Degenhardt, and Surya Kallumadi. 2017. The Architecture of eBay Search. In *Proceedings of ACM SIGIR Workshop on eCommerce, Tokyo, Japan, August 2017 (SIGIR 2017 eCom),* 7 pages.
https://doi.org/

1 INTRODUCTION

Like a traditional web archive, a marketplace such as that at eBay forms a large search space. At the time of writing there were about a billion searchable documents (items) available on eBay. Unlike a traditional web archive, a marketplace such as eBay sees rapid change to that document collection, with approximately 20% of the collection changing every day. Also unlike a web archive, changes in the document collection must be propagated to the users immediately.

An example scenario illustrates this immediacy. An item may be seeing a bidding war where two or more users are pushing up the price, while a third user is searching for a "good deal" listing results to their search ordered on price from low to high. That third user expects the price they see on their screen to be accurate at the moment of the search (despite the bidding war). If the item sells to one of the bidders, then a fourth user should not be able to find that item even if it was sold only a few milliseconds earlier. And it is obvious that two users cannot be permitted to buy the same unique item.

Although this appears to be a simple case of database search, it is not – it is complex Information Retrieval search. eBay buyers can, and do, issue queries along the lines of "blue Converse All Stars", price restricted to under $10, and ordered on price including shipping. Such a scenario requires a free-text search (to find the shoe), a facet restriction (under $10), the use of personalized information about the seller (the shipping from address), the buyer (the shipping to address), and feature data such as the weight of the item (necessary to compute the shipping cost). In this example, the rank order of items is dependent on the seller and the buyer and consequently a different user is likely to see a different ordering making result caching near impossible.

Clearly there is much in common between a traditional large scale web search engine and eBay. Both require distributed search in order to search the collection quickly. Both require replicated search in order to maintain high levels of reliability. Both require multiple data centers to mitigate network latency issues and to insure against network or power problems.

In this position paper we outline the current state of the eBay search engine (called Cassini), and outline the reasons behind some of the engineering decisions. We start with the search engine itself, before moving on to briefly discuss ranking (unfortunately the details of which are proprietary), we then discuss the indexing pipeline. Finally we introduce one of the many difficult and unique eCommerce challenges eBay faces: updates to sellers of items.

2 DISTRUBUTED IR

There is a practical limit to the number of queries that can be serviced on a single machine in a given time period. Results from the SIGIR RIGOR .gov2 evaluation showed the top performing system, JASS [8], resolving queries in about 28ms – or only 36 queries per second on short queries. A system like eBay must scale in three dimensions, to increased queries per second, to increased complexity of those queries, and to an increased number of documents searched as the size of the document collection grows. The natural solution is a distributed architecture.

The usual distributed solution is to break the document collection into several equal sized chunks called shards, and to search each shard on a different CPU core (or machine) and in parallel. The machines that resolve the query are referred to as query nodes and results are then merged at a broker.

The number of shards that are needed is a function of the query load and the number of documents. Replication is used to address an increase in query rate. This results in a grid of machines where columns are shards and rows are replicas. eBay takes exactly this approach with the exact configuration changing over time (by manual re-configuration) in order better address the behavioral changes of the users, advances in hardware, and a growing document collection.

It is relatively easy to show that the same document will be found regardless of the sharding strategy. However, it is also relatively easy to show that for any ranking function using IDF it is not possible to guarantee that the relative rank order of the documents will remain the same if the shards are searched in isolation then merged. Prior work suggests several solutions to this [3] including keeping a central vocabulary of terms and frequencies seen across all the shards. In a search scenario undergoing rapid change (such as eBay), keeping a central vocabulary is impractical. A better solution comes from uncooperative distributed search: probing [10]. With probing the search engine first sends a query to a query node in order to ascertain term statistics, the query is annotated with those statistics and then distributed back to the query nodes to be resolved. Using this approach the global statistics are computed on the fly and IDF is, consequently, correct at the moment of query processing – ideal for a rapidly changing collection. In practice, probing is expensive as it substantially increases network traffic. It also introduces an additional level of dependence between the broker and the query nodes as the query cannot be resolved at any query node until the broker has fully annotated the query.

eBay, like many distributed search engines, prefers the central nothing approach. That is, the task of the broker is to distribute the query to the query nodes and to merge the results, while the task of the query node is to search its shard. In this way the query nodes are independent and there is no cross-system dependency. Although the global term statistics necessary to correctly compute IDF are inaccurate, the statistics stored at each node are a good-enough estimate of the global statistics because each shard is large (typically tens of millions of documents).

There has been considerable research into selective search (also known as topical sharding) [6] and collection selection by using source selection approaches such as ReDDE [11]. It is unclear from the literature whether topical sharding is used in any commercial search engine, and it is not used at eBay. One consideration is failure tolerance. Random sharding appears to offer better resilience to query node drop-out than topical sharding, as it naturally avoids dropping large numbers of relevant documents. Also importantly, query streams are topically unpredictable – especially after a world event. For example, when a famous musician dies eBay can see a peak in queries for merchandise and memorabilia of that individual. With random sharding that increased load is spread across all the query nodes, but with topical sharding a single node might be expected to resolve the increased query load on its own – leading to an increase in latency and a decrease in quality of service. Recall that caching is difficult in a system like eBay due to the high rate of change in the document collection (and individualized results listings) – and so all queries pass to all query nodes to be resolved. The cache does not and cannot compensate for the increased query traffic as there is no cache.

The task of the broker (referred to, at eBay, as an aggregator) is to simply receive a query from further up the protocol stack, to distribute that to the query nodes, then to receive the results from the query nodes, to merge them, and to pass the merged list back up the protocol stack.

eBay supports several different datasets that a user can search. The most obvious is known as Active Item, those items currently for sale through the site. But it is also possible to search through those items that have recently been sold, known as Completed Item. To allow the searching through either Active Item, Completed Item, or the two combined, a second aggregator (the Top Level Aggregator) is used. This aggregator examines the query to determine which set of documents should be searched, sends the query to the appropriate set, aggregates the results and sends that back up the protocol stack.

Figure 1 Illustrates the topology of the eBay search engine. Queries eventually arrive at the top-level aggregator which then distributes them to the correct lower level aggregators (based on collection to search) which then distributes to query nodes to do the search in a shared nothing (central nothing) architecture. Top-k results are then passed up the tree, each time being merged at an aggregator before being returned back to the user. Other elements of Figure 1 are discussed in other sections.

3 RELIABLE IR

Any production environment expected to remain up 24 hours a day 7 days a week is faced with possible software, network, or hardware failure. This is often addressed through replication. In the case of eBay, a data center is divided into a logical grid where columns are shards (query nodes) and rows are replica sets. Various software components are constantly monitoring the state of the grid and reporting any failures.

These replicas are a normal part of the data center – they are all resolving queries as those queries arrive. But some queries take longer to resolve than others. A software load balancer is used to ensure replica sets do not form bottlenecks. Each time a query arrives the software load balancer identifies the most idle replica set and forwards the query to that replica set.

Should a replica set fail, the software load balancer stops sending queries to that set until it eventually returns. Should an individual node in a column fail then the low-level aggregator knows how to fail over to a “spare” replica of that individual node. This same approach was discussed by Barroso et al [1] with respect to the Google search engine.

As with any large-scale web search engine, eBay has a sufficiently large number of computers in the grid that it is reasonable to expect that at any moment in time at least one is in a failed state. Various systems identify this, try and bring those nodes back up, try to reboot machines, and after going through a checklist, notify a hardware engineer of a fault.

Once a node returns to service it rejoins its replica set by notifying the broker of its reincarnation. The broker will then send queries to that node. This is also illustrated in Figure 1 which shows two lower level aggregators with exact replica sets of the query nodes for the Active Item collection.

4 STRUCTURED IR

By default, the user queries on the eBay site search only the listing titles – users can, however, specify that a search should search not

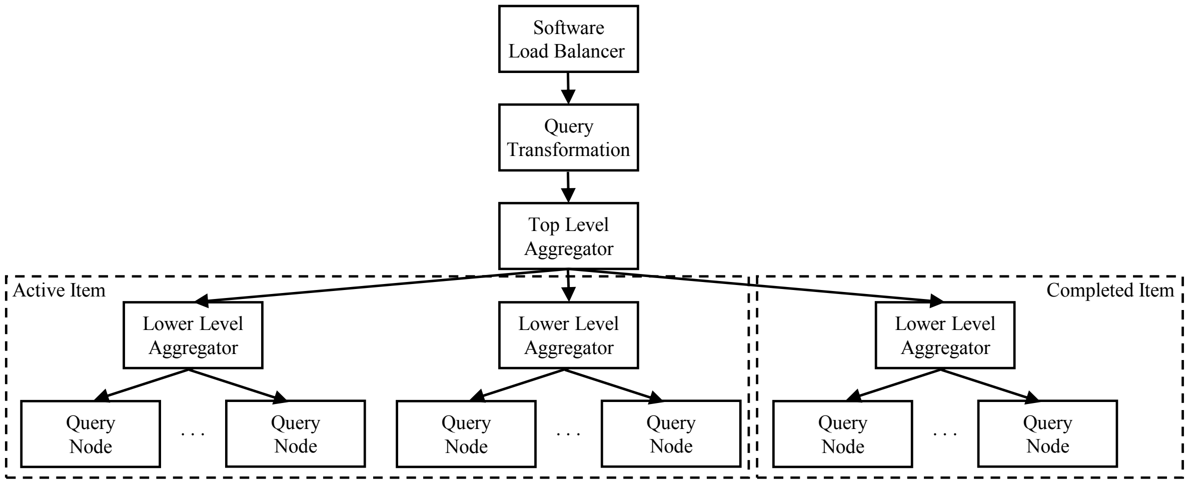

Figure 1: Two levels of aggregator (broker) talking to query nodes (search engine instances) makes it possible to search active item or completed item or both, all in parallel. However, before any of this load ballancing and query transformation occurs.

only the title, but also the description of the items. This is an example of structured (or fielded) information retrieval. Surprisingly, there does not appear to be a single best index structure for structured search. Trotman [12] suggests building a single DOM-like super-tree over all documents, numbering the nodes, and annotating postings lists with node numbers. Guo et. al. [5] stores a Dewey encoded path directly in the postings lists. Both these approaches involve annotating each posting in a postings list.

Clarke et. al. [2] introduce region algebra, and in doing so show that the structural information can be stored separate from the postings lists. If the postings lists encode word positions from the start of the collection (rather than document id and term frequency) and if the document start and end positions are stored in a separate list then it is possible, given any posting, to determine which document it came from. If a document contains several different structures then the start and end of these structures can be stored in lists, one for each structure. Given a term restriction (e.g. “Converse in Description”), the search engine loads the postings for the term, filters the list using the field’s positional information, and converts the result into a document id. This approach is particularly applicable if the document collection is XML and the task is focused retrieval [13].

Figure 2 illustrates region algebra. It shows two documents (items) in XML, the term positions, and the regions are shown at the starts of each element. Given a term position and the regions it is possible to determine whether or not the term lies in the region. For example, “red” occurs at position 7, and the title regions are 1-3 and 5-6 so “red” does not occur in the title of any document.

eBay sometimes uses region algebra albeit encoded differently from Figure 2. As the task is document retrieval (not focused retrieval), the postings lists store document id, term frequency, and word positions within that document. The region lists encode document id, field frequency, and the start and length of that field in that document. There are several reasons for this encoding, the most obvious of which is that by storing <start, length> pairs the

Collection	Position	Region
<item>		1-4
<title>		1-3
Converse	1	
All	2	
Star	3	
</title>		
<description>		4-4
Blue	4	
</description>		
</item>		
<item>		5-7
<title>		5-6
Low	5	
Tops	6	
</title>		
<description>		7-7
Red	7	
</description>		
</item>		

Figure 2: A collection of 2 structured documents, the term positions and the regions (start and end positions) used for region algebra.

difficulty of resolving self-containing regions (e.g. a <b> element inside a <b> element) is diminished. Another reason is efficiency. When a query contains a field restricted phrase it can be resolved by first resolving the phrase query then resolving the field restriction. Region algebra, although provably correct, does not scale well for semi-structured document collections. For a field that occurs many times in a document (such as <p> in an HTML document) the lists that contain the start and end positions of that field can become very long – longer than the postings lists for even the most frequent terms. Processing these long lists takes time and can overwhelm the overall processing time of a query. This does not happen within eBay because documents are structured (not semi-structured) and so structures occur at most once per document.

An obvious and far simpler approach to structured information retrieval is to build an inverted index over each separate field in the collection. This approach also does not scale, this time if there are a large number of fields within a document. If the user is able to choose any combination of fields to search over then each inverted list for each field must be examined – which is computationally expensive.

The eBay indexer builds separate inverted indexes over a small number of fields as well as an index over the entire document, this time without word positions. A user is then able to restrict a query term to one of these fields or to the entire collection, but never to more than one of these (and they cannot use phrases). This approach has the advantage of decreasing the complexity of a field restricted search because the number of postings in a field is typically smaller than in the entire collection, and smaller again because word positions are not stored.

Figure 3 illustrates the two documents from Figure 2 each with two fields (title and description) sharing a single vocabulary. If the user searches for "star" anywhere in the collection then the postings for "star" are examined. If the user searches for "star" in the title then the postings for "title:star" are examined. If the user searches for "star" in the description then, since there is no "description:star" in the vocabulary, there is a vocabulary mismatch and no documents will be found. Recall that field restricted phrase search can be performed using region algebra.

Term	Postings
all	<1,1>
blue	<1,1>
converse	<1,1>
low	<2,1>
red	<2,1>
star	<1,1>
tops	<2,1>
title:all	<1,1>
title:converse	<1,1>
title:low	<2,1>
title:star	<1,1>
title:tops	<2,1>
description:blue	<1,1>
description:red	<2.1>

Figure 3: The fielded index for document collection in Figure 2, assuming each document is separated by <item> tags. Postings are represented <docid, tf>.

5 QUERY TRANSFORMATION

When a query arrives at the eBay search engine it is immediately sent to a number of subsystems that are responsible for re-writing it in several different ways. These query transformations can be either explicit or implicit. In explicit query transformation, such as spelling correction, the user sees the transformed query. In implicit query transformation, the user does not. Search engines use query transformation as a way to increase the recall of queries while trying to minimize any negative effect on precision. But it is an especially critical aspect of eCommerce search as it affects the purchasing behavior of the user. If the transformation results in items that are not relevant then it reflects badly on the site. Equally, if the transformation results in retrieving too few items then it affects revenues and reduces the purchasing options for the user.

Query transformation often happens through a process of query understanding then query rewriting. The objective of query understanding is to identify the meaning of a query and incorporate that into the result retrieval process. For example, for the query "2006 Golf", the search system should understand that the search intent is more likely to be cars and less likely to be sports.

The objective of the rewrite is to encapsulate user intent so as to retrieve a different set of relevant results to rank. This is especially useful when there is a long tail of items. Rewrite is achieved in two ways 1) Query expansion and 2) Query relaxation. Query expansion broadens the intent of the query by adding additional relevant tokens. Query relaxation removes tokens from the query, the objective of query relaxation is to make the queries less restrictive thus increasing recall. Of course, a query might have all its terms relaxed and then be expanded with a completely new set of terms.

Query suggestion is another important aspect of all modern search engines. At eBay, the objective of query suggestion is to explicitly nudge the user to construct a query that is aligned with their intent. Query suggestions are created by mining the query logs and looking at reformulations. The transient nature of products at eBay adds additional complexity. We do not want to suggest queries for items that are no longer available because doing so results in a negative user experience.

5.1 Automatic Category Navigation (DSBE)

The behavioral data generated by users searching and buying items on eBay is a valuable source of query disambiguation data (amongst other things). For example, if a user enters "bread" as a query, eBay knows, from that behavioral data, that the user is more likely to be interested in bakeware than music. This knowledge is mined from the query logs.

When a user enters a query it is logged, when they click on items, that too is logged. Eventually, when the user purchases an item that is logged. In an eCommerce environment it is more difficult to determine this click chain that in a web environment. A web session is often considered to be a continuous sequence of requests terminating after a period of about 10-15 minutes of user inactivity [4]. However, in an eCommerce environment a user might search for an item (for example, a car), watch that item for a period of time, periodically returning to examine characteristics of the item (for example, the type of stereo), compare that item to several others, then eventually purchase it (or not). The determination of a session on an eCommerce site is beyond the scope of this contribution.

Periodically the query and click streams are mined. One purpose of the mining task is to determine, for the most frequent queries, how to disambiguate those queries. The difficulty within eBay is that any user can list almost any item for sale, and many of these are rare or one-off items – so it is often not feasible to associate frequent queries with items via a product catalog alone. However, it is often feasible to use associations with structured data components, such as category, brand, etc. that can be further associated with items. The eBay query disambiguation engine takes this approach.

One of the more valuable associations is the most likely category that will satisfy the query. For the example, “bread” is more often bakeware than music so the results list is limited to just the bakeware section of eBay. Another is any special considerations on how to rank the query. That is, the query itself carries details of the ranking function to be applied by the search engine.

5.2 Query Rewrite (SIBE)

A more complex mining task is that of determining user intent rather than simply disambiguating the query. This, too, is mined from user behavior logs and the document collection. For each frequent query a set of purchases and a set of non-purchases forms a dataset that is analogous to a relevance feedback [9] dataset.

It is clear from relevance feedback experiments [14] that a new query can be formed from the original and the labeled data, and that query will better fulfills the user’s information need than the original query. At eBay these rewritten queries are stored keyed on the user’s original query and a quick lookup returns the replacement query.

Examination of these re-written queries shows that they tend to include boosting of synonyms and stemming variants, but also down-ranking of ambiguous and noise terms in order to clarify the query. For some queries these re-writes can be large in size – not infrequently exceeding hundreds of kilobytes. Unfortunately, these long queries take a considerable amount of the computational resources to resolve, a necessary trade off in order to successfully fulfill the user’s needs.

5.3 Profile Lookup Service (PLS)

The eBay ranking functions are stored externally from the main source code of the search engine. They are often generated through data mining and machine learning. For example, a forest of gradient boosted decision trees is learned and used at the top level of a multi-stage ranking (i.e. learning to rank) function.

There is no reason why these trees should be constant across queries, users, or locations – and they are not. There are commonly many different ranking functions in the system simultaneously. They may serve different search applications or different query segments within the same search application. A/B tests are an especially important case. To perform an A/B test of a ranking function its necessary to “tie” the user to a particular function for the duration of the test.

These capabilities are provided by the Profile Lookup Service. It allows eBay search to define and house many different ranking functions. It allows them to be retrieved by search application, user (for A/B tests), and other dimensions of the search environment. It also serves to isolate front-end applications from the details of the recall and ranking algorithms.

6 RANKING

Although we are not at liberty to release details of the eBay ranking function, some details are either standard practice or already released.

By default, only the titles of the documents are searched. However, the queries are re-written to include two parts: first, those terms useful in determining a suitable set of documents to rank (the recall base); and second, those terms used in ranking. In this way terms that are necessary to identify a relevant document (but frequent in that set of documents) can be ignored in ranking, and terms that are frequent in the collection but not in the recall base can be used for ranking. These terms are mined from user logs.

Multi-stage raking is used. The final stage uses a forest of gradient boosted decision trees to perform the final ordering. These trees (and lower levels too) draw from textural features of the document (such as term proximity), as well as features the user has supplied (such as price). They are deduced from the customer and seller (location information used for postage computation), seller reputation, and over 500 features eBay stores about an item (ranging from color to condition to size). Much of this feature information is stored in a forward index (or docdata).

Unusual in academic search but common in industrial search, eBay search avoids returning no results whenever possible. If a query is about to return no results, or a small number of results, then the null-and-low subsystem is notified. There are several reasons recall might be low; for example it might be due to the absence of products in the repository (the query is highly specific and the item does not exist) or due to a misalignment of vocabulary between the user and the product (for example, “apple tablet” rather than “iPad”).

The null-and-low component takes the query, re-writes it to a new query with similar semantics, and sends it back to the search engine. This process repeats until either a suitable number of results is identified, or the process gives up. We cannot release the details of the null-and-low subsystems, however a user can observe the eBay spelling checker at work if a single nonsense word is entered. They can also observe the dropping of query terms if a nonsense word and a non-nonsense word are entered as the query. Section 5.1 introduced automatic category restriction for queries. Experiments showed that this has several effects on the search engine. First, by reducing the size of the recall base the query latency goes down. Second, by removing noise from the results list the precision goes up.

7 INDEXING

Maintaining an inverted index over a rapidly changing document collection has received little attention in the past. Indeed, it is often assumed that indexing efficiency is unimportant as indexing only occurs once. This is not the case at eBay or with any other rapidly changing collection that must be kept up-to-date (for example a news site).

Lester et al. [7] offer three update strategies for maintaining an inverted index, in-place, re-merge, and re-build. The in-place approach accumulates changes to postings lists in memory then merges those into a master index, keeping the postings list at the same location if it still fits – and if it does not then it appends it to the end of the index. Re-merge simply merges the existing index with a set of diffs to create a new index. Re-build discards the old index and re-indexes everything from scratch. Their experiments were on-disk and so the efficiency results are not applicable to an in-memory index.

At eBay a combination of approaches is used. Every 8 hours a complete re-build of the gold-standard document collection is performed. This "bulk" is then shipped to the index serving grid, trickling from row to row so that, over time, the entire grid has a fresh index. On a more frequent basis (minutes), all changes to the gold-standard document collection are computed and shipped to the grid as a "mini" index. These minis contain newly indexed documents and a list of deleted documents (document changes are deletes followed by adds). There is, clearly, a short period when both bulks and minis are being built at the same time. This is resolved at the query nodes and in distribution where it is ensured that indexes arrive in the correct locations and in the correct order.

The in-place index merging approach of Lester et al. is used to combine bulks and minis, except that the index resides in memory not on disk. One at a time each postings list that requires appending is pulled from memory. If it contains a posting for a deleted document then that posting is removed from the list, then the new postings are added to the end. The postings list is put back in memory in-place if it fits and if not then elsewhere in memory. The consequence of this approach is that postings lists that are not added to are also not purged of deleted documents. There are several reasons for choosing this approach. One is that it is not necessary to exactly re-parse the document in order to determine which postings lists need purging. A second is that it reduces fragmentation because fewer lists are modified.

As a postings list might contain a posting for a document that has been deleted it is also necessary to keep track of deleted items. Documents that have been deleted between the construction of the mini and the time of the search are removed from the results list further up the protocol stack.

Each merge of a mini results in a little more memory fragmentation as some postings lists no-longer fit back in memory where they came from. This is one of the key reasons for an 8-hour bulk re-indexing. Each time a bulk index is loaded the memory on the query serving node, that node is defragmented as the new bulk index lies consecutively in memory and all old patches can be purged. Other approaches are possible, for example, a periodic process could move postings lists about in memory in a de-fragmentation pass.

Another important and pragmatic role of bulk indexes is to provide a vehicle for large updates affecting many documents, but that do not have freshness requirements offered by mini index. Distributing large updates via bulk indexes has strong efficiency benefits.

The 8-hour bulk reload also ensures that three times a day all machines in the grid are synchronised. When a new machine is added to the grid it simply needs to wait for the bulk to arrive to be in sync.

If a machine fails (for whatever reason) and comes back on-line, it can simply compare time stamps of which indexes are stored locally against a global registry of indexes to work out what needs to be done to re-sync – and that registry need not be concerned with indexes prior to the previous bulk.

An alternative strategy of keeping up-to-date is to build the index on the query serving node in real-time as changes occur. This approach results in a substantial increase in the amount of network traffic seen at the query serving nodes. It is also difficult to re-sync in the case of node failure. That is, even if real-time changes are maintained at each node, a master index of that node must be build and maintained so that it can be distributed for a restart.

The index building process is, itself, relatively straightforward. A large HBase cluster is used as the gold-standard document collection. That no-sql database is built and maintained through a series of processes that listen to the eBay internal item messaging pipeline for messages about changes to items and sellers, and updates are applied to HBase accordingly. Although not the topic of this contribution, different processes in different parts of eBay are responsible for determining different characteristics of an item; for example, it is automatically classified, spam detection algorithms are applied, language translation occurs, and more. Updates to a document can occur at any time and HBase ensures these updates are applied in the correct order even if they arrive out of order. This HBase database is not the eBay-wide gold-standard (that is elsewhere), it is the search department's gold-standard.

HBase is built on top of Hadoop, the Apache open-source map-reduce platform. Periodically a map-reduce process kicks-off, the indexer is shipped to the data, indexing occurs in a distributed manner, then the index parts are reduced to form a number of indexes that matches the topology of the query serving grid. These are then picked up by the index distribution agent and moved to the query serving nodes.

8 SELLERS, ITEMS, INDEXES, RANKING

The ranking function draws from many hundreds of different features. Those features are derived from the item, the seller, and the buyer. Features of the buyer are determined high in the protocol stack and are constant for each item. Features of the item are stored in the inverted index and forward index. It is not clear where to store the features of the seller in this eCommerce search problem.

If the seller features are stored in HBase along with the item then any change to a seller would require a change to all items

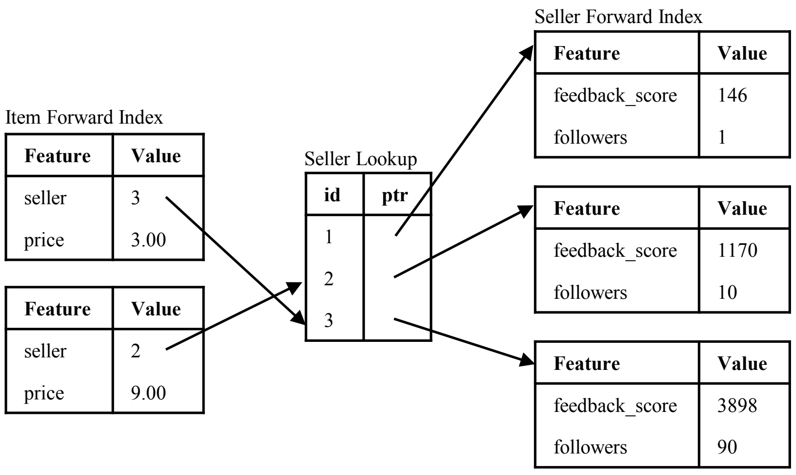

Figure 4: Seller data is stored in a forward index pointed to indirectly from the item forward index.

that seller is selling – which can be many hundreds of thousands of items (especially for sellers of Compact Disks). If the seller data is "crossed" with the item data at indexing time then a change to the seller data would require a substantial update to the index with a mini. Such a change in seller data might occur if, for example, the seller declares that they are on holiday and so the expected shipping date of all their items is to be postponed a short time.

As part of the indexing process eBay builds two indexes, one for the items and one for the sellers. Each item has a seller id associated with it, but the reverse it not true (to find all items of a given seller, an item search for the seller's id is performed). To extract the seller features during ranking a seller lookup is performed on the seller's id and the seller's doc-data is examined.

Figure 4 illustrates this process. The document ids from the inverted index point to the item doc-data (forward index) which contains a seller id which is used to find the seller forward index which contains the seller's features used in ranking.

The seller index only contains data for sellers who have items for sale at the moment the index is constructed. The alternative is to include all potential sellers, but since all users are potential sellers this table would be large and contain many unused entries (which is space inefficient). It is clear that the closure of the set of current sellers can be constructed from the closure of the set of items, and indeed that is how it is constructed.

9 CONCLUSION

In this position paper we outlined much of how the eBay search engine currently functions along with the engineering reasons for it working in this way.

Although many standard techniques are used (inverted indexes, learning-to-rank, etc.), it is not possible to simply take an off-the-shelf web search engine and apply it to eCommerce as other standard techniques (caching, index-once philosophy) are simply untenable. This, along with the rapid rate of change are the reasons eBay chose to build a search engine – and this paper outlines many of the engineering decisions taken.

REFERENCES

[1] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. 2003. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro* 23, 2 (March 2003), 22–28.

[2] Charles L. A. Clarke, G. V. Cormack, and F. J. Burkowski. 1995. An Algebra for Structured Text Search and a Framework for its Implementation. *Comput. J.* 38, 1 (1995), 43. https://doi.org/10.1093/comjnl/38.1.43

[3] O. de Kretser, A. Moffat, T. Shimmin, and J. Zobel. 1998. Methodologies for distributed information retrieval. In *Proceedings. 18th International Conference on Distributed Computing Systems (Cat. No.98CB36183)*. 66–73.

[4] Ayse Goker and Daqing He. 2000. Analysing Web Search Logs to Determine Session Boundaries for User-Oriented Learning. In *AH*.

[5] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. 2003. XRANK: Ranked Keyword Search over XML Documents. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*. ACM, New York, NY, USA, 16–27.

[6] Yubin Kim, Jamie Callan, J. Shane Culpepper, and Alistair Moffat. 2017. Efficient Distributed Selective Search. *Inf. Retr.* 20, 3 (June 2017), 221–252.

[7] Nicholas Lester, Justin Zobel, and Hugh E. Williams. 2004. In-place Versus Re-build Versus Re-merge: Index Maintenance Strategies for Text Retrieval Systems. In *Proceedings of the 27th Australasian Conference on Computer Science - Volume 26 (ACSC '04)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 15–23.

[8] Jimmy Lin and Andrew Trotman. 2015. Anytime Ranking for Impact-Ordered Indexes. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval (ICTIR '15)*. ACM, New York, NY, USA, 301–304.

[9] J. J. Rocchio. 1971. Relevance Feedback in Information Retrieval.

[10] Milad Shokouhi, Falk Scholer, and Justin Zobel. 2006. Sample Sizes for Query Probing in Uncooperative Distributed Information Retrieval. In *Proceedings of the 8th Asia-Pacific Web Conference on Frontiers of WWW Research and Development (APWeb'06)*. Springer-Verlag, Berlin, Heidelberg, 63–75.

[11] Luo Si and Jamie Callan. 2003. Relevant Document Distribution Estimation Method for Resource Selection. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval (SIGIR '03)*. ACM, New York, NY, USA, 298–305.

[12] Andrew Trotman. 2004. Searching Structured Documents. *Inf. Process. Manage.* 40, 4 (May 2004), 619–632.

[13] Andrew Trotman, Nils Pharo, and Miro Lehtonen. 2007. *XML-IR Users and Use Cases*. Springer Berlin Heidelberg, Berlin, Heidelberg, 400–412.

[14] Andrew Trotman, Antti Puurula, and Blake Burgess. 2014. Improvements to BM25 and Language Models Examined. In *Proceedings of the 2014 Australasian Document Computing Symposium (ADCS '14)*. ACM, New York, NY, USA, Article 58, 8 pages.